Database Systems
CSE 344

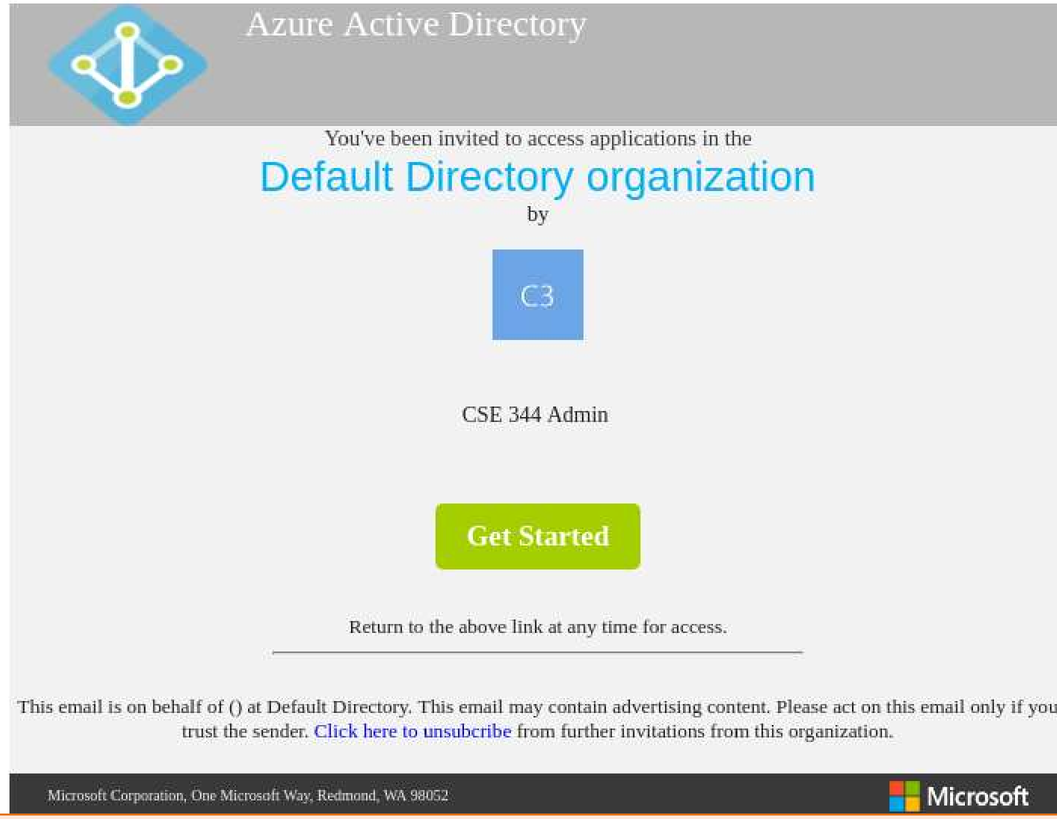
Lectures 9: Relational Algebra (part 2) and Query Evaluation

(Ch. 5.2 & 16.3 (skim 16.3.2))

Announcements

- WQ3 is due on Monday July 10
- HW1 grades were be posted
- HW3 - Get Azure Setup

Azure Subscription



You will get an email like this to your Azure account.

Relational Algebra Operators

- Union $\cup$, intersection $\cap$, difference $-$
- Selection σ (Sigma)
- Projection π (Π) (Pi)
- Cartesian product $\times$, join $\bowtie$
- Rename ρ (Rho)
- Duplicate elimination δ (Delta)
- Grouping and aggregation γ (Gamma)
- Sorting τ (Tau)

RA

Extended RA

Join Summary

- **Theta-join**: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  - Join of R and S with a join condition θ
  - Cross-product followed by selection θ
- **Equijoin**: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  - Join condition θ consists only of equalities
- **Natural join**: $R \bowtie S = \pi_A(\sigma_{\theta}(R \times S))$

  R.id = S.id
  - Equijoin
  - Equality on **all** fields with same name in R and in S
  - Projection π_A drops all redundant attributes

More Joins

- **Outer join**
  - Include tuples with no matches in the output
  - Use NULL values for missing attributes
  - Does not eliminate duplicate columns

- Variants
  - Left outer join (⟕)
  - Right outer join (⟖)
  - Full outer join (⟗)

More Examples

size	color
15	red
17	Green

$\Downarrow$

15, red
15, red
17, Green

Supplier(<u>sno</u>,sname,scity,sstate)

Part(<u>pno</u>,pname,psize,pcolor)

Supply(<u>sno</u>,<u>pno</u>,qty,price)

Name of supplier of parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10}\ (\text{Part})))$

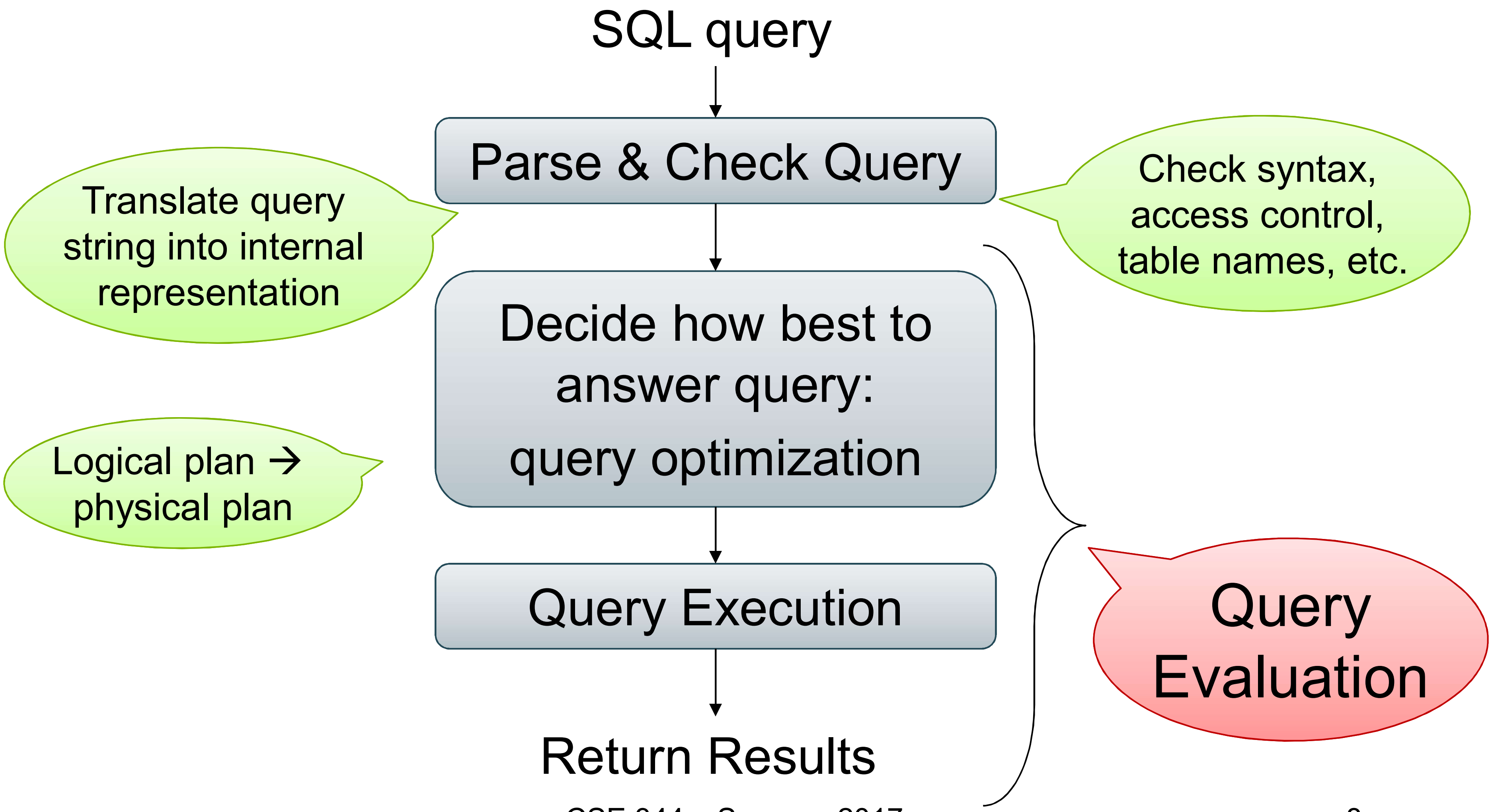
Name of supplier of red parts or parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10}\ (\text{Part}) \cup \sigma_{pcolor=\text{'red'}}\ (\text{Part})\ )\ )$

$\bowtie \approx \cap$

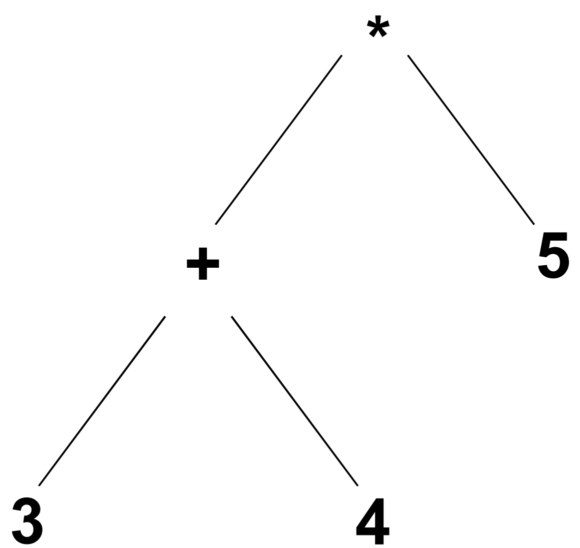
$\sigma_{psize > 10 \vee pcolor = \text{'red'}}$

Query Evaluation Steps

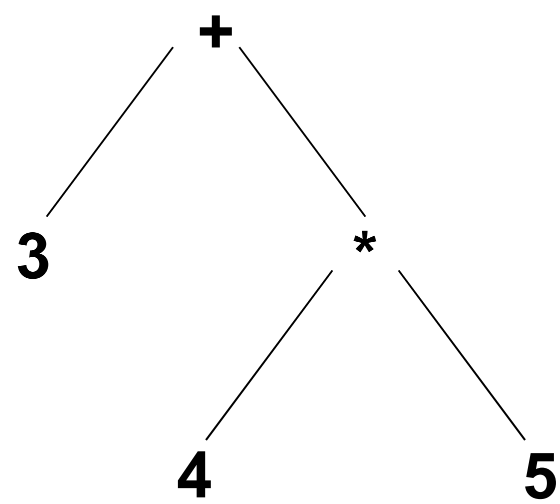


From SQL to RA

- SQL → RA → Syntax Tree
- Graphical representation of evaluation order.



(3 + 4) * 5

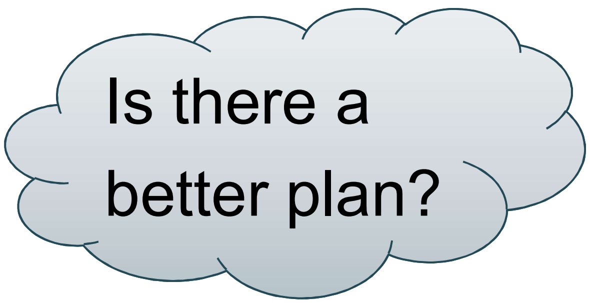


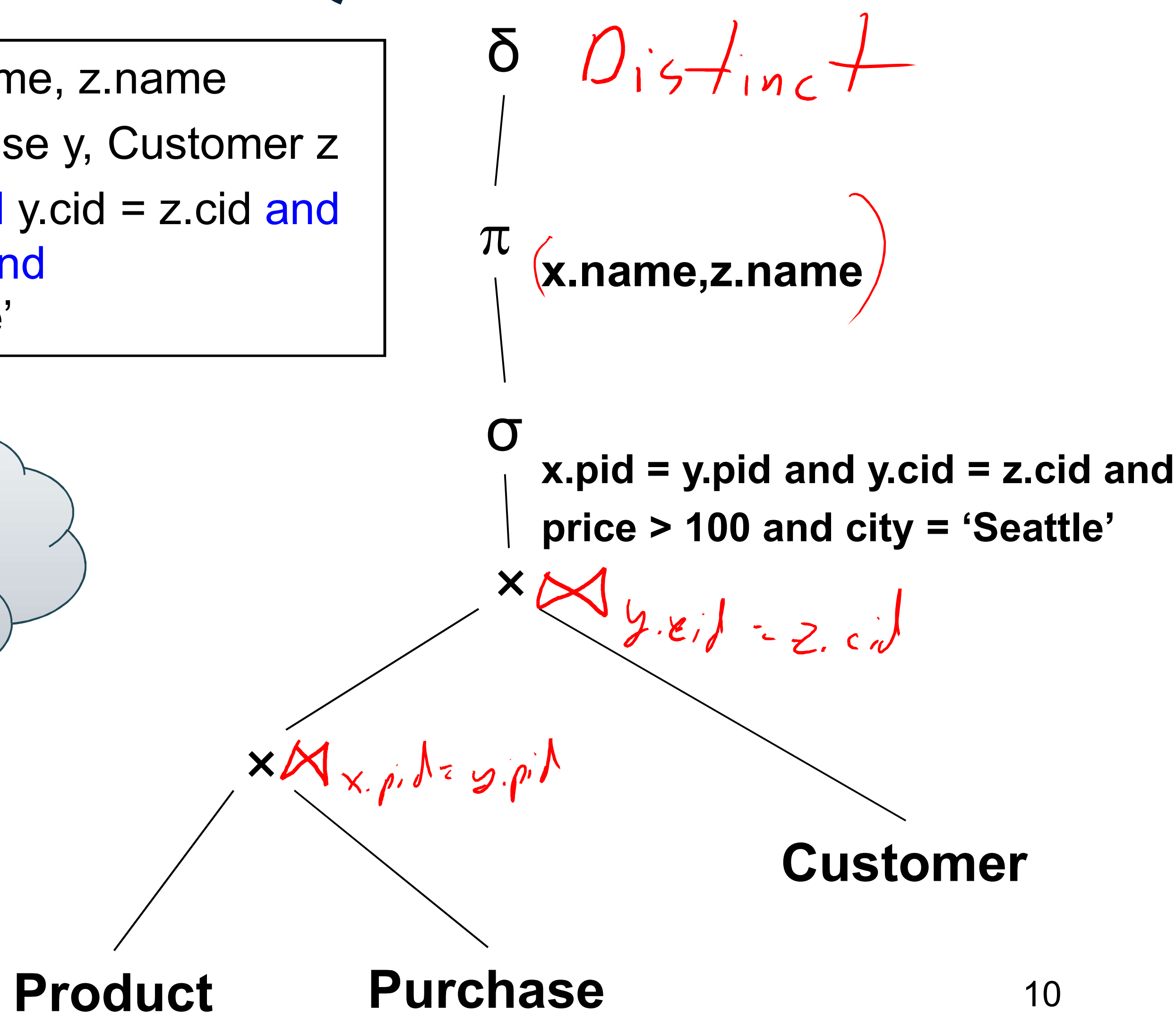
3 + 4 * 5

Product(<u>pid</u>, name, price)
Purchase(<u>pid, cid</u>, store)
Customer(<u>cid</u>, name, city)

From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
           x.price > 100 and
           z.city =  'Seattle'
```





RA: Equivalence Transformations

Conjunctive selection operations can be deconstructed

$$\sigma_{\theta 1 \wedge \theta 2}(E) = \sigma_{\theta 1}(\sigma_{\theta 2}(E))$$

Selection operations are commutative

$$\sigma_{\theta 1}(\sigma_{\theta 2}(E)) = \sigma_{\theta 2}(\sigma_{\theta 2}(E))$$

Thata joins are commutative

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

Natural joins are associative

Thata joins sometimes

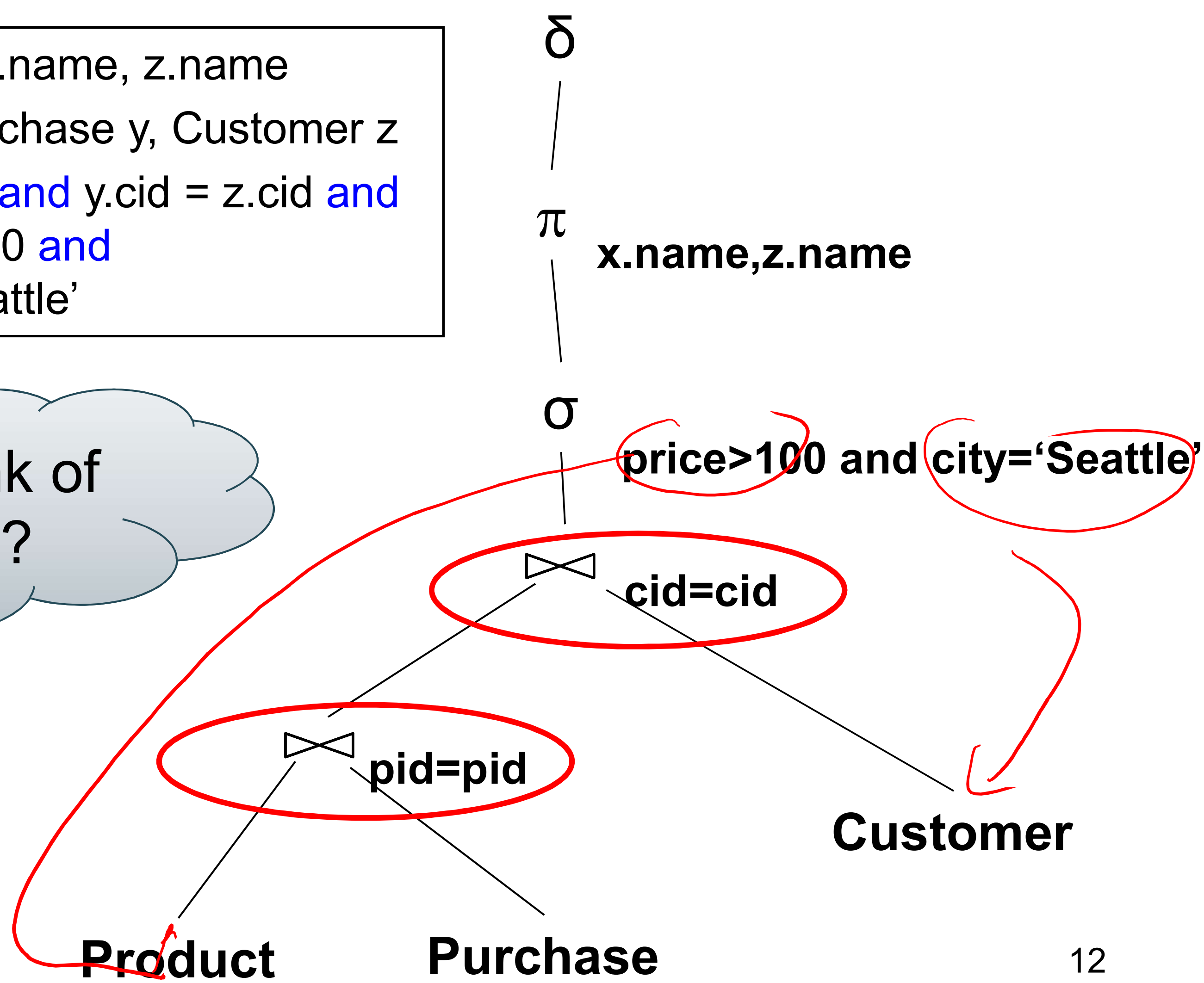
$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
      x.price > 100 and
      z.city = 'Seattle'
```

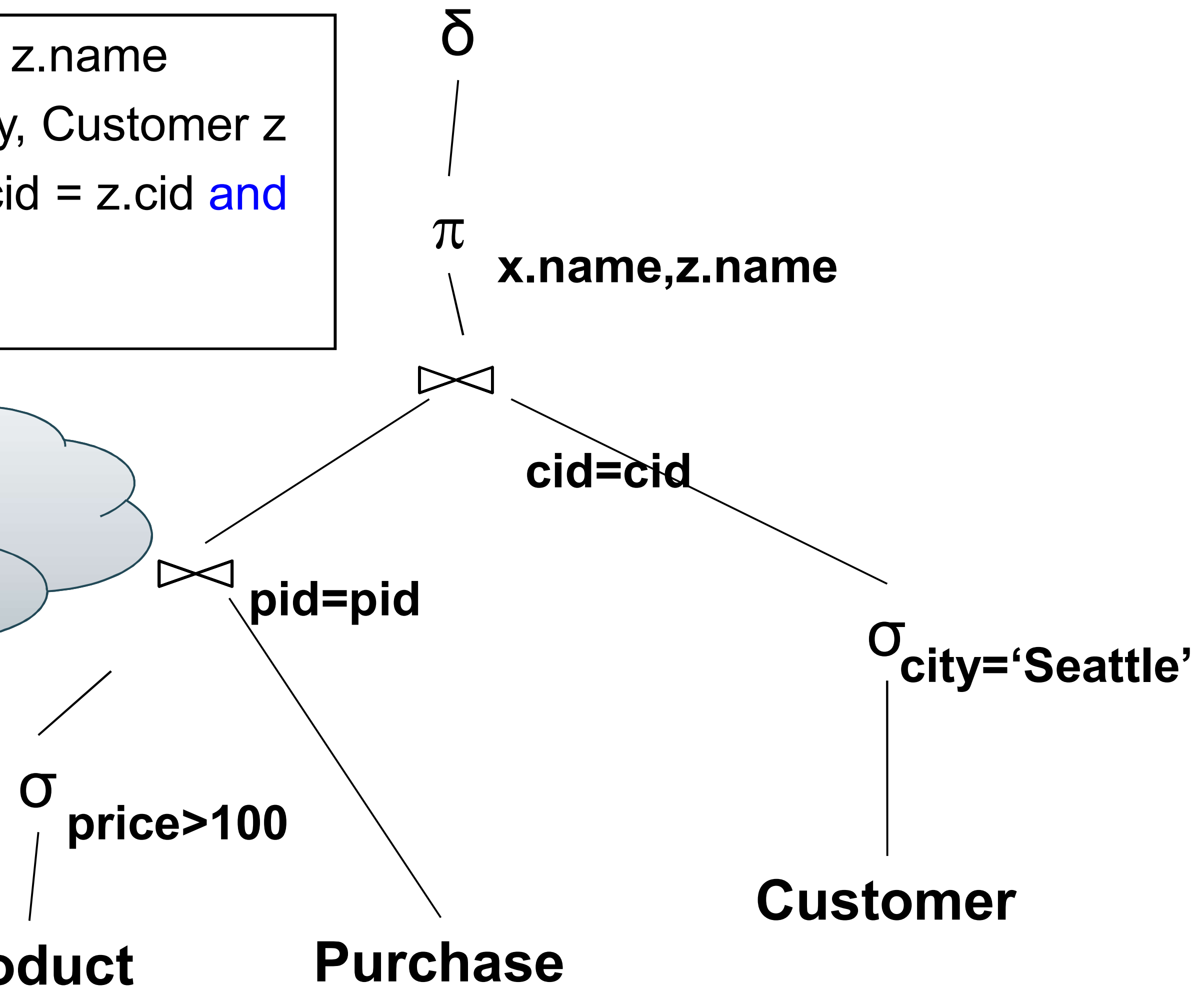
Can you think of another plan?



Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
          x.price > 100 and
          z.city = 'Seattle'
```



Can you think of another plan?

Push selections down the query plan!

Query optimization: find an equivalent optimal plan

Extended RA: Operators on Bags

- Duplicate elimination δ
- Grouping γ
  – Takes in relation and a list of grouping operations (e.g., aggregates). Returns a new relation.
- Sorting τ
  – Takes in a relation, a list of attributes to sort on, and an order. Returns a new relation.

Using Extended RA Operators

```sql
SELECT city, count(*)
FROM sales
GROUP BY city
HAVING sum(price) > 100
```

T3(city, c)

$\Pi_{\text{city, c}}$

T2(city,p,c)

$\sigma_{\text{p > 100}}$

T1(city,p,c)

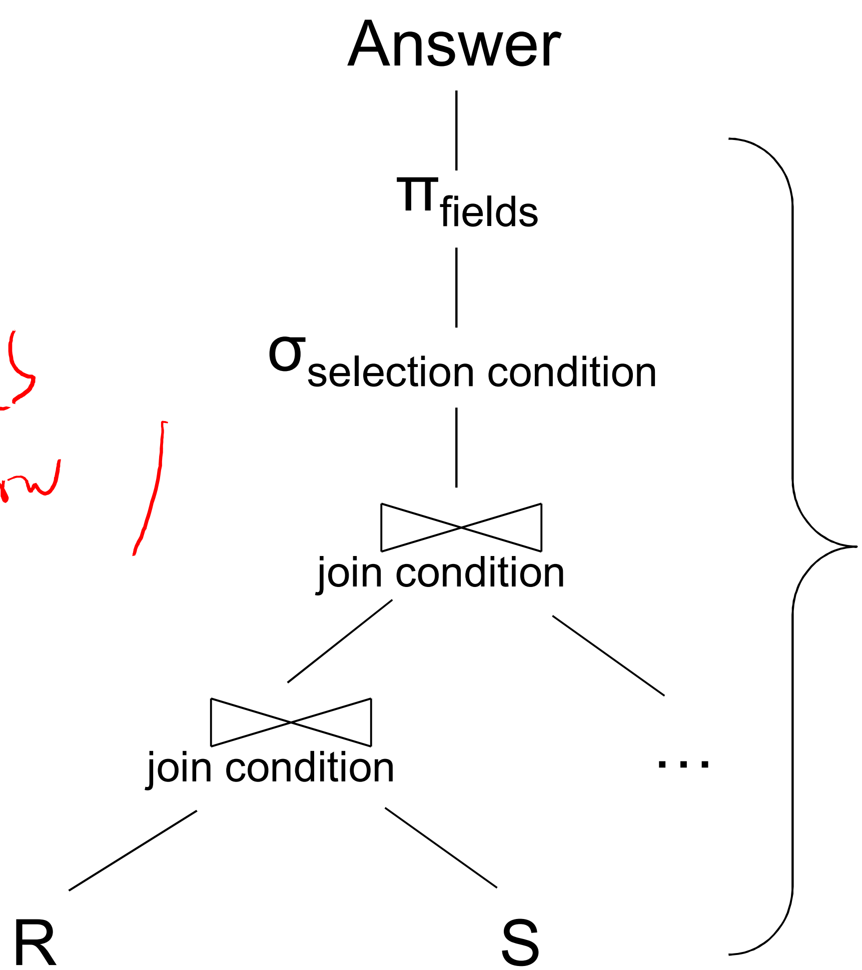
$\gamma_{\text{city, sum(price)} \rightarrow \text{p, count(*)} \rightarrow \text{c}}$

sales(product, city, price)

T1, T2, T3 = temporary tables

Typical Plan for a Query (1/2)

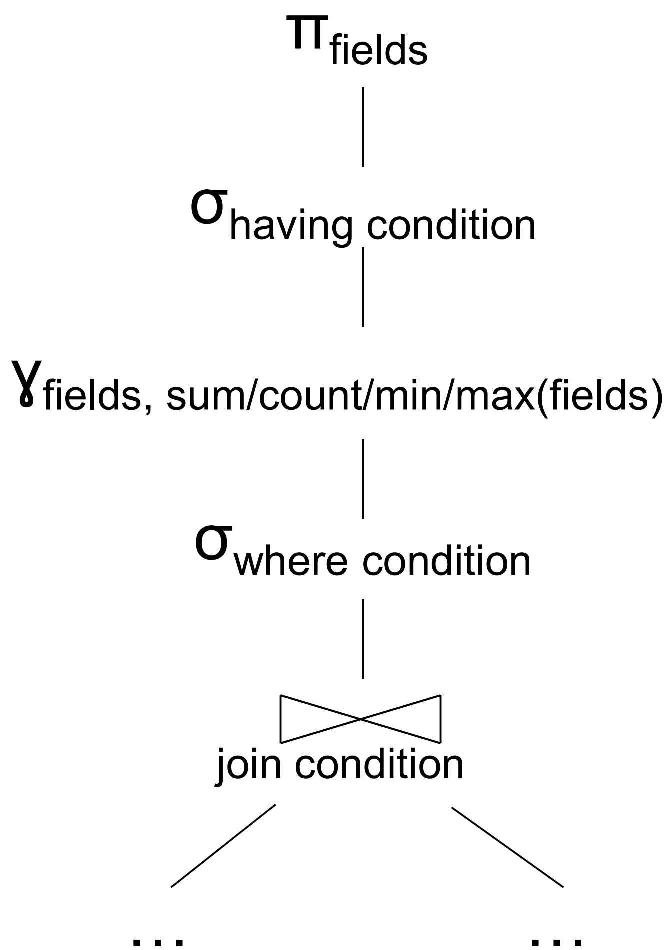
Push down




```
SELECT fields
FROM R, S, …
WHERE condition
```

SELECT-PROJECT-JOIN
Query

Typical Plan for a Query (1/2)




```
SELECT fields
FROM R, S, …
WHERE condition
GROUP BY fields
HAVING condition
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
    and not exists
      (SELECT *
       FROM Supply P
       WHERE P.sno = Q.sno
           and P.price > 100)
```

Correlation !

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
    and not exists
      (SELECT *
       FROM Supply P
       WHERE P.sno = Q.sno
           and P.price > 100)
```

De-Correlation

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
    and Q.sno not in
      (SELECT P.sno
       FROM Supply P
       WHERE P.price > 100)
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

How about Subqueries?

Un-nesting

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = 'WA')
   EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

EXCEPT = set difference

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
    and Q.sno not in
        (SELECT P.sno
         FROM Supply P
         WHERE P.price > 100)
```

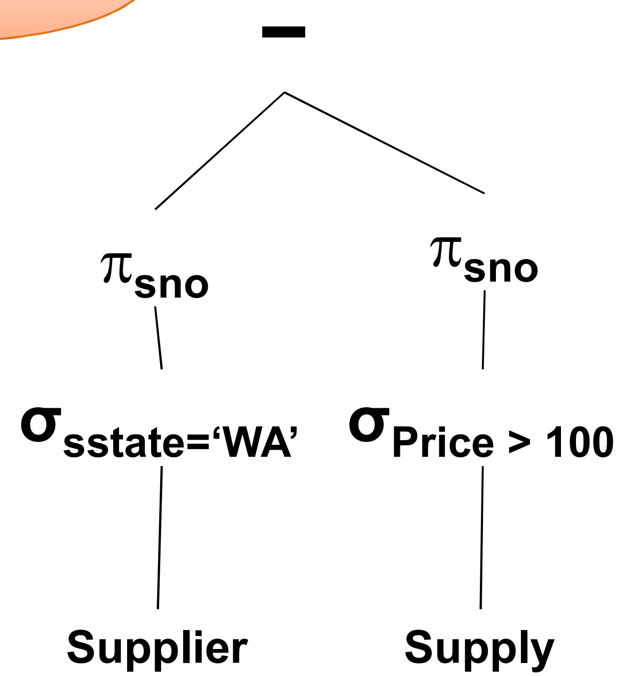
no RA

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)

How about Subqueries?

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = 'WA')
   EXCEPT
 (SELECT P.sno
  FROM Supply P
  WHERE P.price > 100)
```

Finally…



From Logical Plans to Physical Plans

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

Relational Algebra

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

Give a relational algebra expression for this query

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Relational Algebra

```sql
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

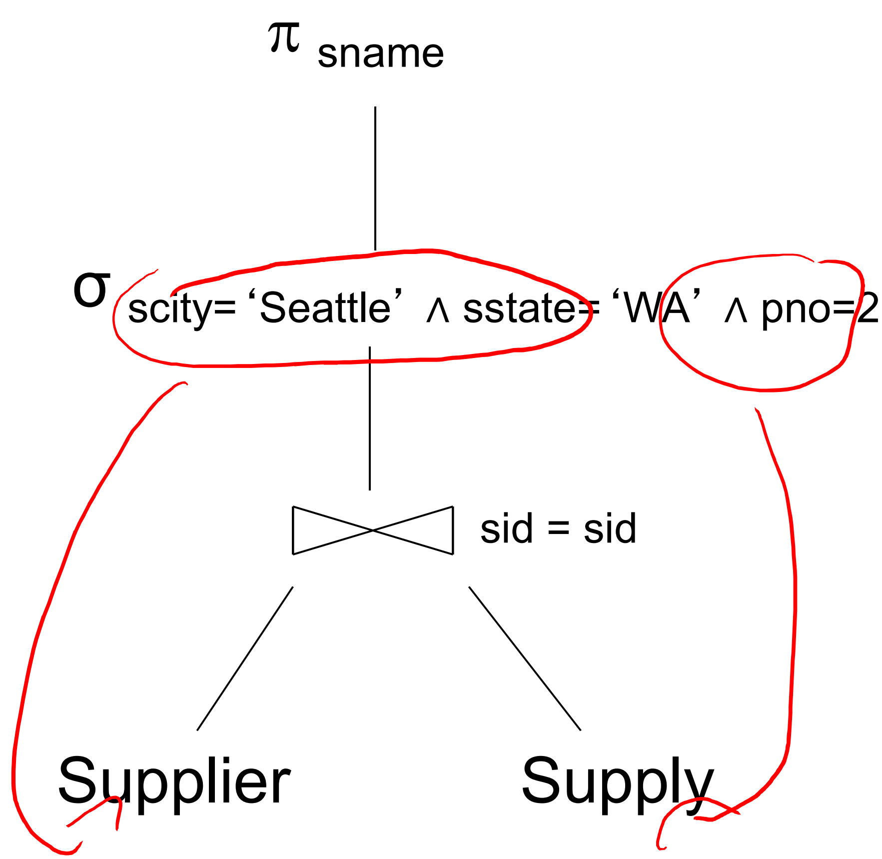
$$\pi_{sname}(\sigma_{scity=\text{'Seattle'} \wedge sstate=\text{'WA'} \wedge pno=2}(\text{Supplier} \bowtie_{sid = sid} \text{Supply}))$$

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

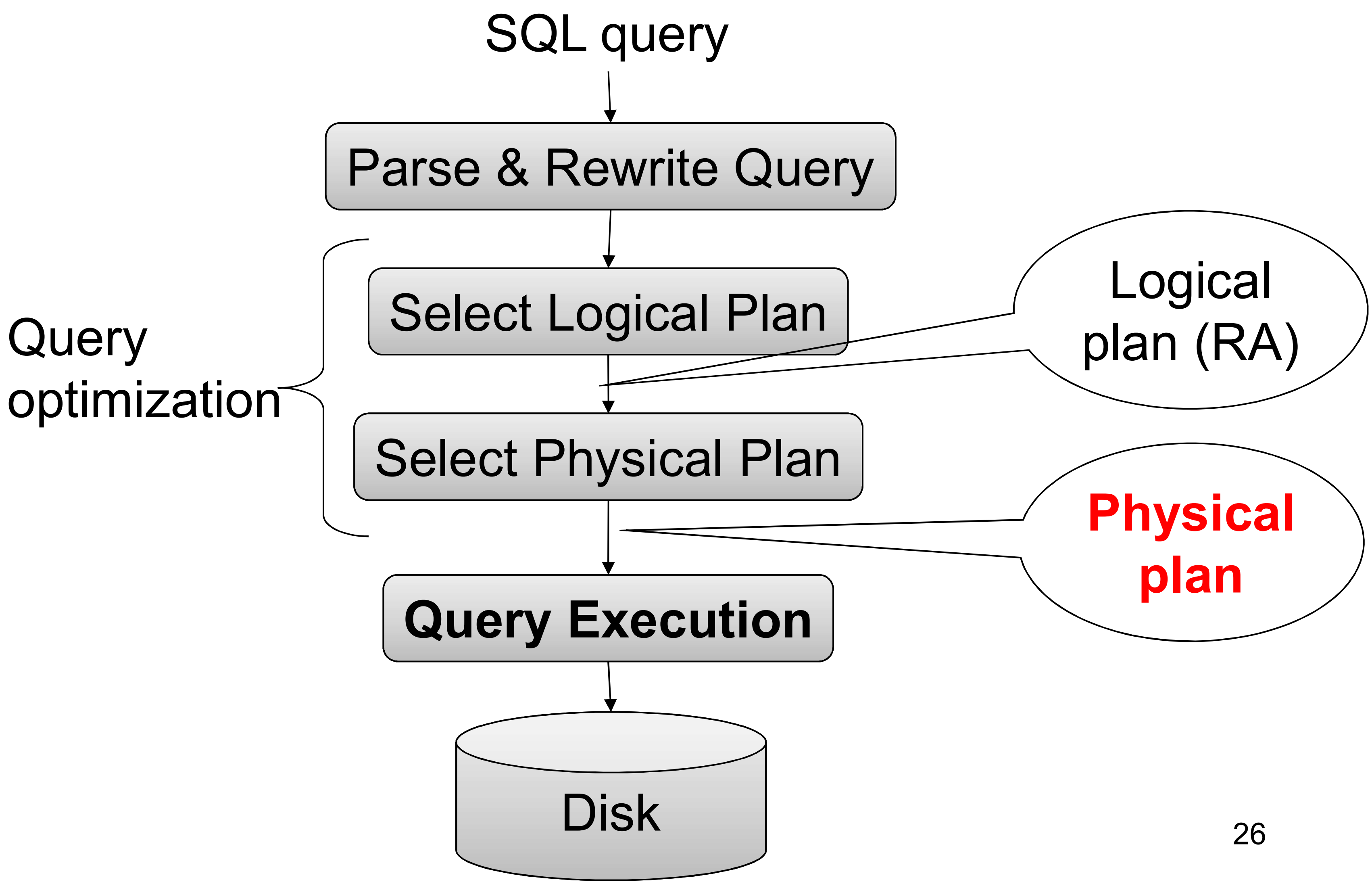
Relational Algebra

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity =  'Seattle'
   and x.sstate =  'WA'
```



Relational algebra expression is also called the "logical query plan"

Query Evaluation Steps Review



Physical Operators

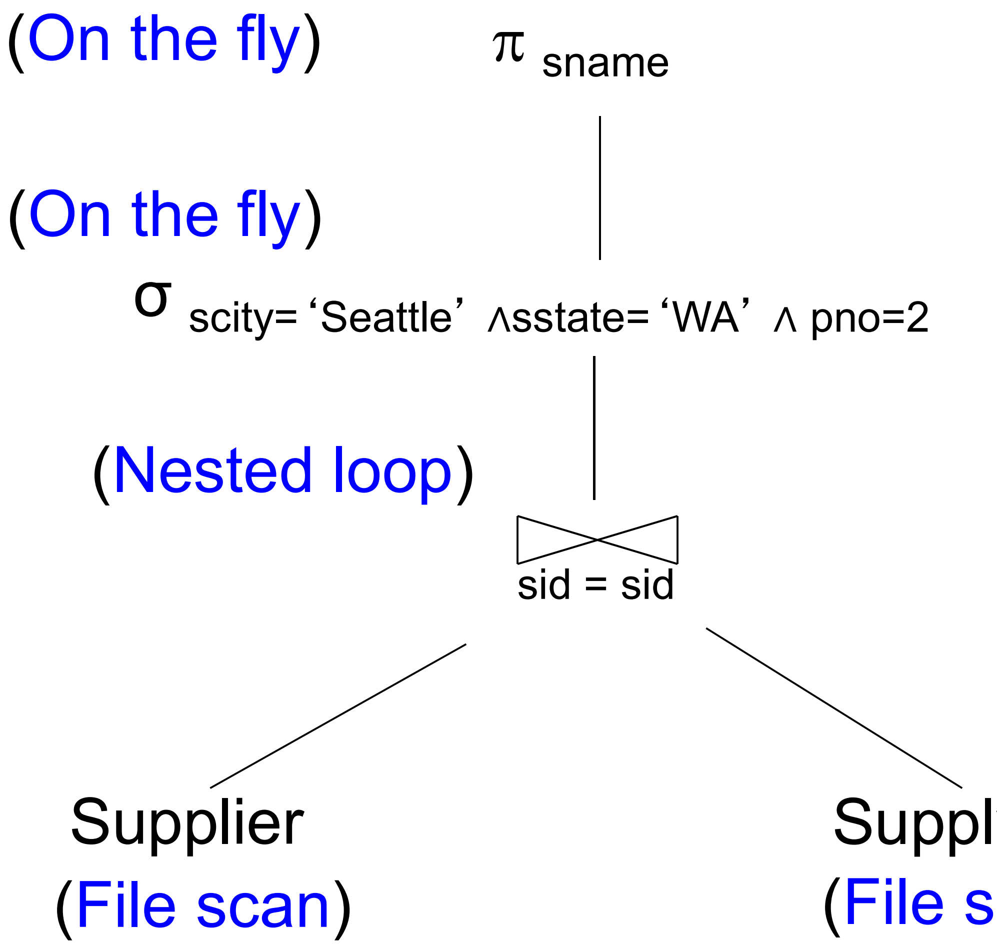
Each of the logical operators may have one or more implementations = physical operators

Will discuss several basic physical operators, with a focus on join

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Physical Query Plan 1



A physical query plan is a logical query plan annotated with physical implementation details

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

Product(pid, name, price)
Purchase(pid, cid, store)

Main Memory Algorithms

Logical operator:

Product(pid, name, price) $\bowtie_{\text{pid=pid}}$ Purchase(pid, cid, store)

Propose three physical operators for the join, assuming the tables are in main memory:

1. Nested Loop Join $O(n^2)$
2. Merge join $O(n \log n)$
3. Hash join $O(n)$ … $O(n^2)$

add n to hash – O(n)?
lookup n in hash – O(n)?

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Physical Query Plan 2

(On the fly) π_{sname}

(On the fly) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'} \wedge \text{pno=2}}$

(Hash join) $\bowtie_{\text{sid = sid}}$

Supplier (File scan)

Supply (File scan)

Same logical query plan
Different physical plan

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Physical Query Plan 3

Different but equivalent logical query plan; different physical plan

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   and  y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

(On the fly) π_{sname}

(Sort-merge join) $\bowtie_{\text{sid = sid}}$

(Scan & write to T1) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'}}$

(Scan & write to T2) $\sigma_{\text{pno=2}}$

Supplier (File scan)

Supply (File scan)

Query Optimization Problem

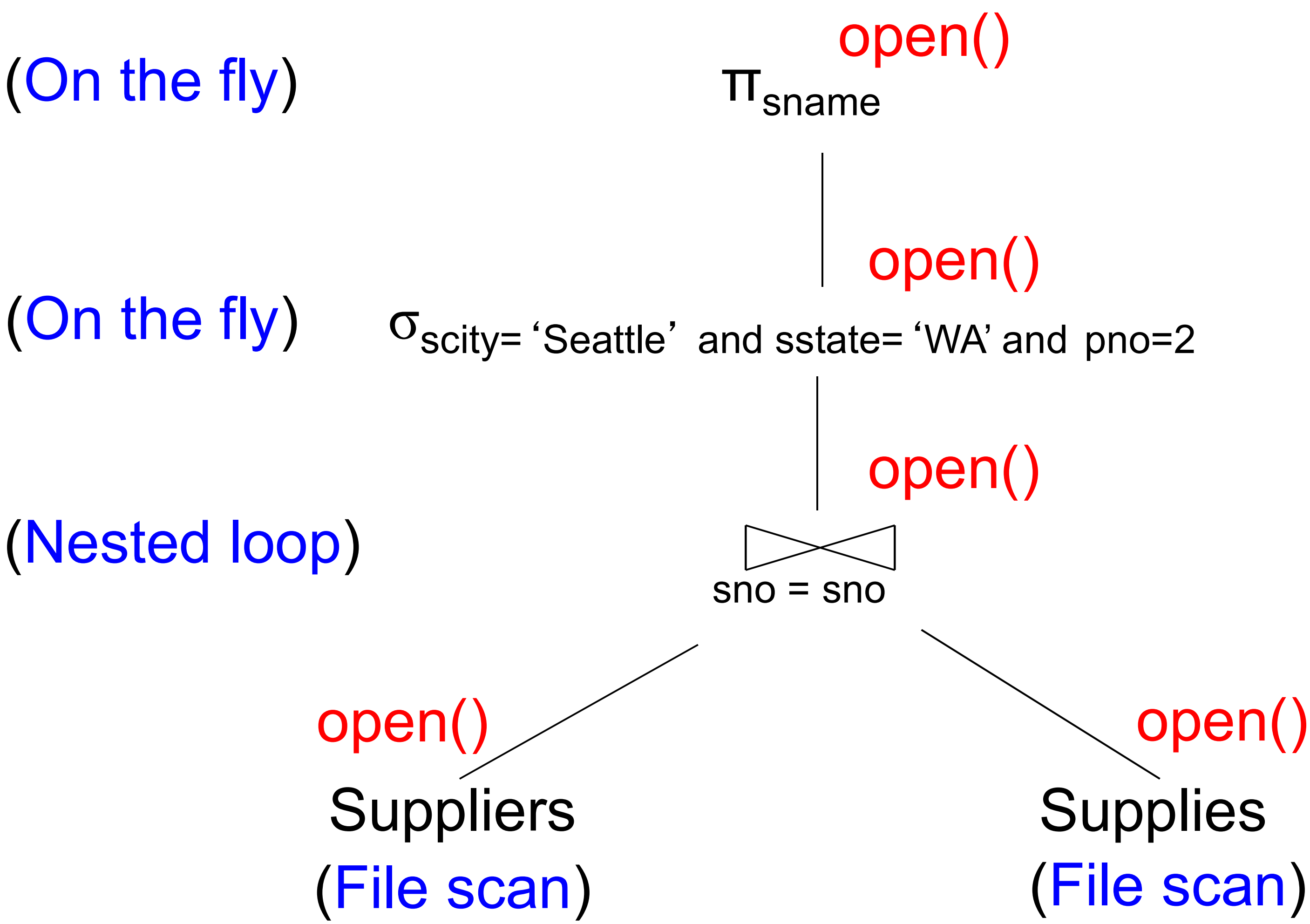
- For each SQL query… many logical plans
- For each logical plan… many physical plans
- How do find a fast physical plan?
  - Will discuss in a few lectures
  - First we need to understand how query operators are implemented

Query Execution

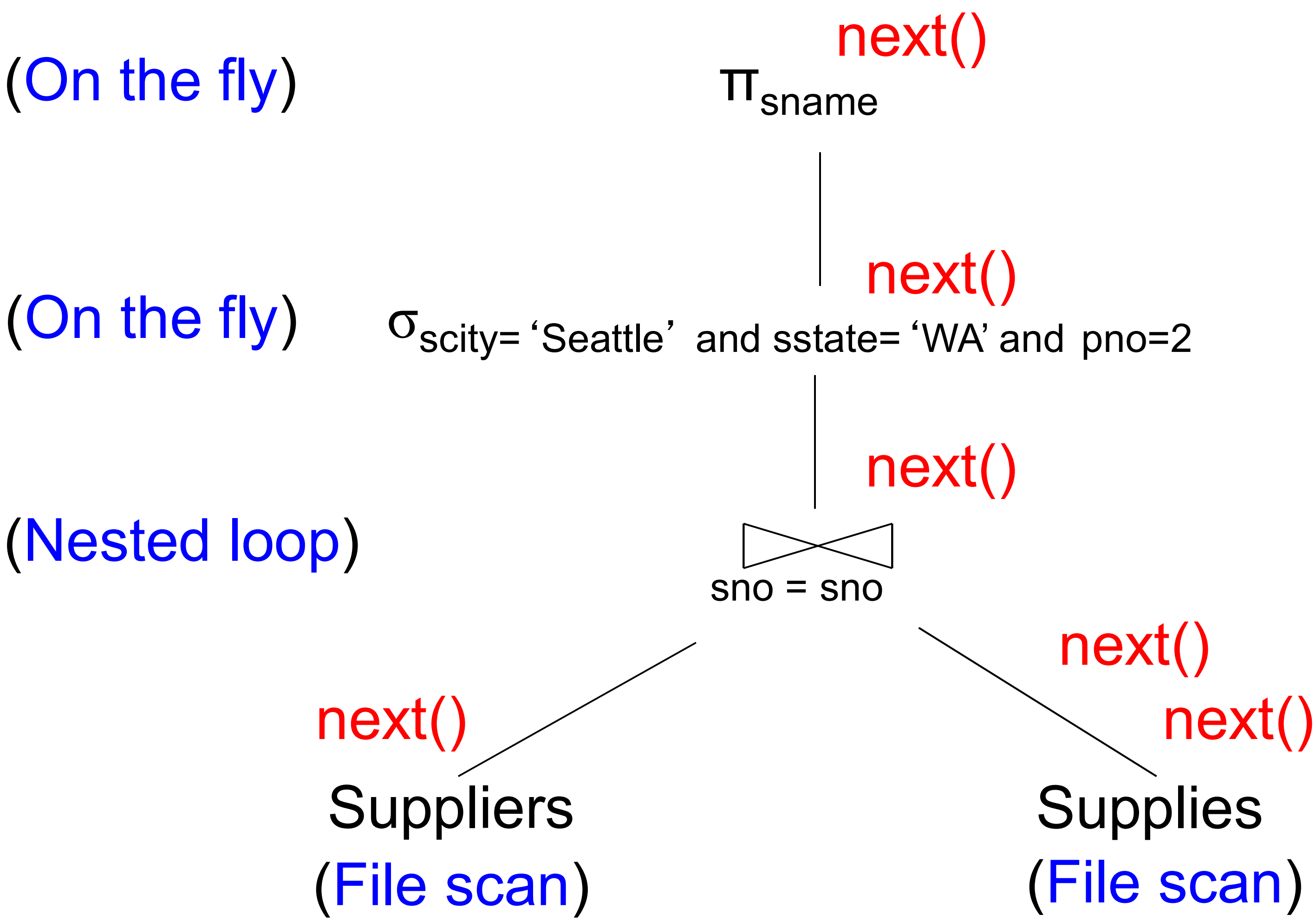
Iterator Interface for Query Operators (Relations)

- **open()**
  - Initializes operator state
  - Sets parameters such as selection condition
- **next()**
  - Operator invokes get_next() recursively on its inputs
  - Performs processing and produces an output tuple
- **close()**: clean-up state

Pipelined Query Execution



Pipelined Query Execution



In 444 you implement this.

Pipelined Execution

- Tuples generated by an operator are immediately sent to the parent
- Benefits:
  - No operator synchronization issues
  - No need to buffer tuples between operators
  - Saves cost of writing intermediate data to disk
  - Saves cost of reading intermediate data from disk
- This approach is used whenever possible

Query Execution Bottom Line

- SQL query transformed into physical plan
  - **Access path selection** for each relation
    - Scan the relation or use an index (next lecture)
  - **Implementation choice** for each operator
    - Nested loop join, hash join, etc.
  - **Scheduling decisions** for operators
    - Pipelined execution or intermediate materialization
- Pipelined execution of physical plan

Physical Data Independence

- Applications are insulated from changes in physical storage details
- SQL and relational algebra facilitate physical data independence
  - Both languages input and output relations
  - Can choose different implementations for operators